Server Side Caching: Distributed Caching or Local Caching with Cache Replication

Eldho Mathulla

December 2017

1 Introduction

One of the easiest way to improve the overall performance of any system is to use caching mechanism, which stores the data with high computational cost temporarily and re used later on.

In the present scenario where we are seeing more a more distributed systems, implementing a caching solution is an interesting challenge. One of the simplest solution here is to use a local cache for each node. But the problem in this scenario is that, if a user tries to access a data through node1, and then after sometime tries to access the same data through node2. In this scenario the data is cached in node1, but not in node2. Therefore in node 2, the data has to be re computed, this caused loosing the effectiveness of caching itself.

To tackle this issue, there are 2 different approaches that can be used:

1. Distributed caching
2. Local caching with cache replication

The main goal here is to evaluate the above mentioned approaches and identify which is a better fit for caching purpose on Apache Airavata API Server.

2 Distributed Caching

For caching in a distributed system, one of the best ways to do so is to use a distributed cache system. This will make sure that the cache can be shared between multiple nodes. In such setup, the idea is to use clusters of node for caching, in which some nodes handle the same cache data replicated across it. i.e. cache replication is happening for a subset on nodes for some specific data. This kind of setup allows to partition the data among nodes in the cluster.

2.1 Implementation Details

For the the implementation of distributed cache, *Redis* was used as the basis.

The main reason behind using redis is that, it allows to partition the data, along with using cache replication and has handled fault tolerance and can scale well. Since Redis stores key-value pair in-memory, this should have improved speed and performance when compared to a disk based solutions.Redis is essentially a key-value pair data store. For the caching, the objects that need to be cached is stored as a key value pair in redis, in JSON format. The eviction policy for the cache is handled by redis, so that we don't have to care of that part explicitly.

3 Local Caching With Cache Replication

Local caching with cache replication is modification of traditional local cache by adding cache replication, so as to solve the issue of using a local cache in distributed systems where cached data in in one node cannot be used by another directly.

The main reason for proposing such an approach is the performance when compared to a distributed cache systems. Since the local cache is embedded within the application, the accessing of the cached data should be significantly faster than a distributed cache systems, which exists independently of the application, which can cause significant overhead in accessing of the cache.

3.1 Implementation Details

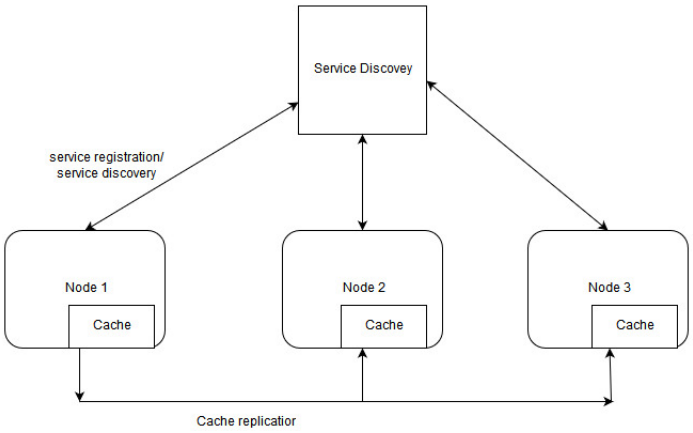
For the implementation of local cache system with cache replication, I have used a local cache called *Caffeine*. Caffeine was chosen mainly because of its extendable nature, which allows to change eviction policy, caching cache store and so forth.

There are 2 important parts to the system: service discovery and cache replication mechanism.

For any sort of cache replication, we need to find other systems, this where service discovery comes into play. When a new node is created ti can register itself so that other nodes can discover it for any cache replication. In this implementation, Apache Zookeeper through Apache curator has been used for service discovery mechanism.

For cache replication mechanism, cache replication should happen at every time when cache is altered in anyway, i.e. cache creation,deletion and modification. To achieve this multiple ways, such as using a queue to using a APIs, can be used . In this implementation I have used APIs based of Restful Web services to provide functionality of cache creation deletion and modification.

When a new node is created, using the cache replication API, it it fills its local cache with cache from one of the existing nodes. When a cache is changed in anyway for one node, the corresponding nodes lets the other nodes know of this changes through this cache replication API.



4 Comparison

While comparing a distributed cache and local cache with cache replication, the advantage of the distributed cache is that, since it is an independent entity, it can be used by any application irrespective of the underlying technology that's being used, which is not possible for a local cache as t heavily tied to the underlying platform used for building the application itself. Another advantage the distributed cache has is the fact the scaling a of cache can be done irrespective of the scaling of application themselves, which is not the case with local cache with cache replication. Any scaling to improve performance of local cache can be only through vertical scaling only.Another problem area is that in distributed cache that has been implemented the cache is available to all the nodes once cache has been written, but for the Local cache with cache replication, it has to wait till the cache has been replicated to the given node. Irrespective of these short comings, the main area where local cache with cache replication has a huge advantage is the cache replication, is the performance of the cache. Since the local cache is embedded in the application, there is little to no overhead for reading and writing from the cache , which should have a significant effect on the performance.

4.1 Performance

For comparing the performance of the Distributed Cache and the Local Cache with cache replication, a performance test was done with 3 nodes with both the setups.The first test was a read only operation with 500 parallel hits on

each node. The second test was a combination of read write operation with 500 parallel hits for each node.

From test1, it revealed for the Distributed cache, the average response time was 29 ms, where as for Local Cache with cache replication was 3 ms. For test2 the average response time for Distributed cache was 1192 ms and for Local Cache replication, it was 771 ms.

The above results shows 80% response time difference for the read-only test and 35% response time difference for the read-write test, between Local cache with Cache Replication and distributed cache. This shows that given necessary resources, a local cache with cache replication performs significantly faster than the distributed counterpart.

5 Conclusion

Given the comparison of the Distributed Cache and Local Cache with cache replication, Distributed cache has upper hand when it comes to scalability, flexibility and availability. But when it comes to sheer performance the local cache with cache replication id far far ahead of the distributed cache. So when it comes to choosing a caches system for the Airavata API server, it comes down to 2 factors: if the total cached data can be stored in a single system Local Cache with cache replication system is the way to go, otherwise the only option is the distributed cache.

6 Resources

1. Github Repository: `https://github.com/eldhomathulla/CachePOC`

7 References

1. `https://en.wikipedia.org/wiki/Distributed_cache`
2. `https://redis.io/`
3. `https://github.com/ben-manes/caffeine`